/Keywords (AAAI, artificial intelligence)

Tracery: Approachable Story Grammar Authoring for Casual Users

Kate Compton, Benjamin Filstrup and **Michael Mateas**
University of California, Santa Cruz
1156 High Street, Santa Cruz, CA 95064

Abstract

We present Tracery, an authoring tool for casual users to build grammars to generate interesting stories. While many modern story generation systems work to maintain narrative causality, generative systems like Racter show that non-causal and even nonsensical systems also have expressive power. Using design principles of direct manipulation and flow, Tracery is designed to allow users to author more stories with greater ease, and fully explore the affordances of grammar-based story generation. A small pilot test of users were able to quickly author engaging stories, and praised the system for its fun and usability.

Introduction

In this paper, we present Tracery, a system designed to enable novices users to author formal grammars for telling stories. Tracery is part of a larger research project to use the principles of flow (Csikszentmihalyi 1997) and direct manipulation (Shneiderman 1983) to design a series of creativity applications for casual users that, in Ben Shneiderman's words "generate glowing enthusiasm among users." These casual creator apps are intended to enable users to produce artifacts that are personally meaningful and easily shareable. Can Tracery, as a creativity support tool for authoring grammars, give users the expressiveness they need to make personally satisfying stories, while still maintaining such a minimal authorial burden that users can primarily write in naturally expressive language instead of abstract metadata?

Formal grammars were chosen because they are very easy for novice users to understand, and straightforward to author. One of this paper's authors, over years of teaching interactive narrative to students, has found that story grammars are a powerful entry point to introduce students to the decomposition of stories into units, a fundamental skill in building narrative intelligences. Grammars also present a clear correspondence between the content being authored and the story being generated, creating a greater sense of directness. Due to the simplicity of a formal grammar expansion system, automated analysis and visualizations can be easily generated from any authored grammar.

The frameworks of direct manipulation and flow each provide sets of clear design principles proven to create engaged and empowered users, which we use to guide our software design. Novice users might never attempt authoring at all in a closed and complex system, but will find novel ways around limitations in a simple system that they enjoy using, showing that "Tasks that could have been performed only with tedious command or programming languages may soon be accessible through lively, enjoyable interactive systems that reduce learning time, speed performance, and increase satisfaction."(Shneiderman 1983)

Related Work

Twine (Klimas 2009), a popular free online implementation of a hypertext narrative architecture, provides a model for building a user-friendly narrative system. Just as Tracery implements the old and well-trodden concept of grammars, there have been many other implementations of hypertext systems before Twine (Short 2012), which have been complicated and closed. Twine's exceptionally approachable interface, focus on individual narrative expression, free and open implementation, and other design decisions "combine to make a uniquely-accessible tool for creating highly personal games, and provide guidance for how platform designers can create tools that similarly encourage this kind of work." (Friedhoff 2013).

Grammars continue to be productive in current research in games design (Dormans 2011), but while grammars were one of the first and simplest architectures for story analysis and generation (Rumelhart 1975), they were soon criticised as being unable to make sophisticated stories, leading Black to suggest that a knowledge-modelling system would be more productive(Black and Wilensky 1979). Later, when Lang implemented a computational grammar for story generation, his system made extensive use of metadata for cause and effect, as "causality and goal-directedness serve as the 'glue' which hold together the states and events which the story comprises" (Lang 1999), and even non-grammar based systems use metadata (Riedl and Young 2010) to maintain causality, a heavy authorial burden, but necessary for these kinds of stories. A simple grammar based approach without metadata could not produce acceptable output:

> "What Mad Libs and Propp's grammar fail to capture is the message level of storytelling. Any storytelling

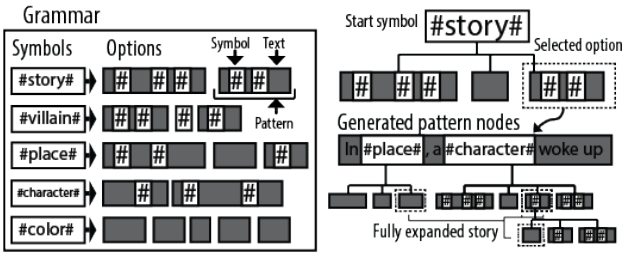

Figure 1: The dictionary of symbols and rules, and the recursive expansion of one symbol into readable text

> system based solely on the surface features of stories– whether a complex system like Propp's or a simple system like Mad Libs–will inevitably fail to be successful." (Turner 1994)

This statement is likely true for many, or even most, use cases of story generation, but there exists a space of creative production in which that requirement does not seem to hold. Not long after Black's critique of grammars, *The Policeman's Beard is Half Constructed*, a book of semi-sensical generated poetry, was released to great interest and acclaim, with one reviewer praising it as "whimsical and wise and sometimes fun" (Nasta 1984), a statement that suggests that, at least for expressive poetry, nonsense was not considered a liability. Siratori's *Blood Electric* (Siratori 2002), a more recent text in the same genre, shows that the interest in computer-generated literature continues. One reason for this is that these works contain enough structure to be readable, sentence by sentence, but lack enough structure to encode enforced expectations, so they generate impossibilities, which, as Bowman describes the joy of MadLibs, forces us to "imagin[e] things as they cannot be" (Bowman 1975). We expect users will find similar pleasure in writing their own surprising generative systems in Tracery.

Design

Tracery is designed with principles of direct manipulation: "visibility of the object of interest; rapid, reversible,incremental actions; and replacement of complex command language syntax by direct manipulation of the object of interest"(Shneiderman 1983) as well as clear and continuous feedback provided by regenerating the story.

User Flow

The user flow is designed to engage the users with fun but limited interactions at first to minimize confusion, then allowing curious users to access increasingly deep and complex customizations, until they find themselves creating a fully personalized grammar to tell their own stories:

- A new user receives a Tracery story from a friend, a page of generated stories from the friend's grammar. More variants can be generated with a single click.
- A selected few customizable rules are initially exposed to the user, allowing easy personalization, like renaming the main character.

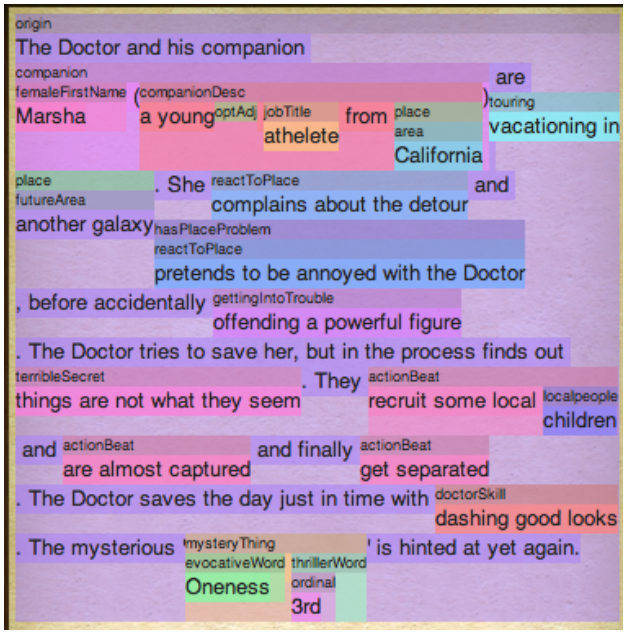

Figure 2: One tester's grammar to generate Doctor Who stories, as shown in the advanced mode. It is clear from the visualization how deeply the symbols of this story are nested

- Eventually the user may open up the story grammar itself to see how it works, and can explore the symbols used in the story.
- The user sees the "add option" button on the symbol, and adds their own text option. The instances regenerate to reflect it.
- With more exploration, the user discovers how to use hashtag and modifiers, and how to add their own symbols, while continuously updating visualizations keep the user aware of how their changes are affecting the possibilities of the story.
- The user sees Tracery generate a particularly amusing instance. The user then proudly shares this instance with their friends as a link (after being prompted to name and save it), and the cycle can begin again with new users.

System Architecture

Tracery implements a standard context-free grammar (CFG)(Black and Wilensky 1979), and extends it with more powerful features that allow the users some additional control while still minimizing tedious metadata authoring.

Tracery itself contains no pre-authored content: the dictionary of symbols Fig. 1 contains only user-generated content.

- Dictionary: a table of all the symbols, either authored by the user, or borrowed from a previous grammar
- Symbol: a unique text string with a number of rewrite rules
- Rewrite rule: a plaintext set of strings or commands for further expansion

Three types of rewrite rules give the user control over the expansion of a symbol: terminals (an unexpandable string), symbols (demarked with hash tags like "#animal#"), and modifiers (preceded with a period: "#animal.capitalize#") to perform simple but desirable transformations of the expanded text, like capitalization or a/an replacements.

These rules are expressed in a simple syntax with hastags, resulting in grammars written expressively in natural language, like the rewrite rule generating the story in Fig. 2:

> "The Doctor and his companion #companion# are #touring# #place#. She #reactToPlace# and #hasPlaceProblem#, before accidentally #gettingIntoTrouble#. The Doctor tries to save her, but in the process finds out #terribleSecret#. They #actionBeat# and #actionBeat# and finally #actionBeat#. The Doctor saves the day just in time with #doctorSkill#. The mysterious '#mysteryThing#' is hinted at yet again."

In the results of the pilot test, users were happy to give up overall story coherence, but did want to have some level of coherence for character names or locations. To implement this, the symbols became stacks of symbols, A fourth kind of rule allows a symbol, in the moment of expansion, to 'push' a new value onto the stack of another symbol, or pop it off. Using stacks to store character and story information neatly encourages the common narrative techniques of frame stories and vignettes, pushing and popping new contexts as substories begin and end, and also fixes CFG's noted inability to maintain a contextual history when a story is interrupted (Black and Wilensky 1979).

Visualization

Writing long combinatorially-complex generative narrative can be confusing for even advanced users, but visualizations can help authors understand the systematic implications of the structures they build, as in Garbe et al's visualization work on Ice-bound (Garbe et al. 2014). In Tracery, novice users mouse over any word in the story to open up the inspector for that symbol. An advanced user can open a nesting colored expansion of the story, showing the details of every symbol hierarchy, color-coded to the original symbol, as in Figure 2. Symbols also show a numerical representation of how likely each symbol is to be used, preventing users from spending too much energy on an unreachable or rarely visited symbol.

Results

This system was implemented and is fully usable, with some features like visualization and networked sharing still in development. It is freely available for any user at: **http://brightspiral.com/tracery/**, and requires only a modern web browser and no additional plugins.

An early version of the system was tested on a 15 person group of computer science and digital art graduate students, who were asked to attempt to make stories and then provide unstructured emailed feedback on their experience. Some testers made very short grammars, others made modifications to the provided grammars, and some students constructed long heavily-nested grammars, showing that already the tool has the potential to overcome the lack of structural variety found in user studies of Wide Ruled (Skorupski and Mateas 2009). As expected, the tool produced absurdity and a perceived lack of self-censorship, which the users found both amusing and useful as a brainstorming tool.

> "It was actually pretty fun. It took a little while to set up everything, but you can get some surprising and pleasantly unexpected behaviour. The generator is clever, sometimes. [...] Also, I'm almost positive I'm going to using the tool for the next few months."
>
> "This was a beautiful brainstorming device,"
>
> "In general, this project is hilarious and relatively simple to use."
>
> "I did find the "mad-libby" style made it a little difficult to make relatable characters and smooth continuity throughout."

As a pilot study, these preliminary results were enough to suggest that yes, people were able to author enjoyable stories using a grammar. The lack of formal causality, considered a dealbreaker in AI-based story generation systems (Lang 1999), did not prevent story creation as prior critiques of grammar-based systems predicted. Users did commonly request a way to maintain consistency of generated properties like names (a feature later implemented with the symbol stack) suggesting that consistency mattered to these kind of stories, even if causality did not.

Conclusion and Future Work

One key to Twine's success was the simplicity of the underlying architecture, which allowed a simple UI to entice novice users. Additionally the ability to export to free and open Javascript/HTML files made it possible for technologically advanced users to extend Twine by using it as a starting point or library, retaining the hypertext-authoring techniques they had honed, while also taking the tool in directions that the creators of Twine could not anticipate. Tracery is designed to follow in Twine's path, and allow a similarly accessible front end, while providing an encapsulated architecture that can be used as an embedded library. Therefore, one next step is to release this encapsulated form so that it can be used in other projects. However, focus will remain on improving the usability of the standalone version.

Tracery is intended to be as usable and accessible as possible, an aim which was encoded at every level of the design of the system. The initial feedback demonstrates that achieving this level usability strongly supported a positive and creative experience, but also that users still saw room for improvement in both usability and flexibility. Since the initial usability so strongly influenced the first test, improving both of these with the new UI and the symbol stack will likely continue to improve the experience.

References

Black, J. B., and Wilensky, R. 1979. An evaluation of story grammars*. *Cognitive Science* 3(3):213–229.

Bowman, D. 1975. Whatever happened to droodles? whatever happened to roger price? *The Journal of Popular Culture* 9(1):20–25.

Csikszentmihalyi, M. 1997. *Flow and the Psychology of Discovery and Invention*. HarperPerennial, New York.

Dormans, J. 2011. Level design as model transformation: a strategy for automated content generation. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, 2. ACM.

Friedhoff, J. 2013. Untangling twine: A platform study. *Proceedings of DiGRA 2013: DeFragging Game Studies*.

Garbe, J.; Reed, A.; Dickinson, M.; Mateas, M.; and Wardrip-Fruin, N. 2014. Author assistance visualizations for ice-bound, a combinatorial narrative. *Foundations of Digital Games*.

Klimas, C. 2009. Twine.

Lang, R. 1999. A declarative model for simple narratives. In *Proceedings of the AAAI fall symposium on narrative intelligence*, 134–141.

Nasta, T. 1984. Book review: Thief of arts. *PC News*.

Riedl, M. O., and Young, R. M. 2010. Narrative planning: balancing plot and character. *Journal of Artificial Intelligence Research* 39(1):217–268.

Rumelhart, D. E. 1975. Notes on a schema for stories. *Representation and understanding: Studies in cognitive science*.

Shneiderman, B. 1983. Direct manipulation: A step beyond programming languages. *IEEE Computer* 16(8):57–69.

Short, E. 2012. Choice-based narrative tools: Twine is epub. http://emshort.wordpress.com/2012/11/10/choice-based-narrative-tools-twine/.

Siratori, K. 2002. *Blood Electric*. Creation Books.

Skorupski, J., and Mateas, M. 2009. Interactive story generation for writers: Lessons learned from the wide ruled authoring tool. *In Proceedings of DAC 09*.

Turner, S. R. 1994. *The creative process: A computer model of storytelling and creativity*. Psychology Press.